CS 520: Assignment 4 - Colorization

16:198:520

Due 12/14 by Midnight

The purpose of this assignment is to demonstrate and explore some basic techniques in supervised learning and computer vision.

The Problem: Consider the problem of converting a picture to black and white.

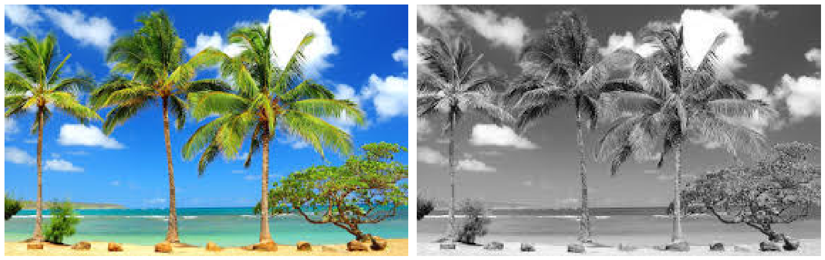

Figure 1: Training Data - A color image and its corresponding greyscale image.

Typically, a color image is represented by a matrix of 3-component vectors, where Image$[x][y] = (r, g, b)$ indicates that the pixel at position (x, y) has color (r, g, b) where r represents the level of red, g of green, and b blue respectively, as values between 0 and 255. A classical color to gray conversion formula is given by

$$\text{Gray}(r, g, b) = 0.21r + 0.72g + 0.07b, \tag{1}$$

where the resulting value Gray(r, g, b) is between 0 and 255, representing the corresponding shade of gray (from totally black to completely white).

Note that converting from color to grayscale is (with some exceptions) *losing* information. For most shades of gray, there will be many (r, g, b) values that correspond to that same shade.

However, by training a model on similar images, we can make contextually-informed guesses at what the shades of grey ought to correspond to. This assignment will walk you through a basic model to accomplish this.

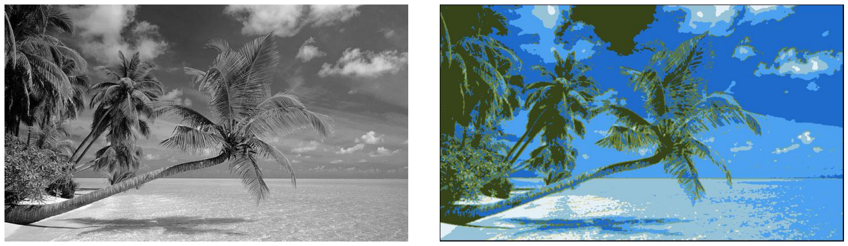

Figure 2: Trained on the Color/Grayscale image in Fig.1, recovers some green of the trees, and distinguishing blues between sea and sky.

Method: Starting with a known color image and its corresponding grayscale version, we will attempt to learn how grayscale content is locally associated with color. As noted, the color-to-gray conversion loses information, so it is not enough to try to learn a correspondence between specific shades and specific colors (though in some contexts, this can be effective). Instead, consider looking at the correspondence between every 3x3 grayscale pixel 'patch', and the 'true' color of the center pixel. This provides a surrounding 'context' (for instance, local texture) that provides more information about the color of the center pixel.

Each line of **input.csv** contains a comma-separated sequence of nine numbers between 0 and 255, corresponding to a 3x3 pixel patch in the known grayscale image. The first three values correspond to the gray values in the top row of the patch, the middle three numbers the middle row of the patch, and the last three numbers the last row of the patch. Each line of **color.csv** contains three comma-separated values, representing the rgb-value of the true color of the center pixel of the patch on the corresponding line of **input.csv**. Using this data, we will construct a map that takes a 9-component sequence of gray values, and suggests a color for the center pixel of the corresponding patch,

$$\text{Color}(\text{gray}_{1,1}, \text{gray}_{1,2}, \text{gray}_{1,3}, \ldots, \text{gray}_{3,3}) = (r, g, b). \tag{2}$$

Once you have constructed such a map, coloring a new gray image is as simple as looking at every 3x3 pixel patch in the gray image, and computing via the map what the color of the center ought to be.

You are welcome to use whatever method you like to try to fit the input data to the color data. The rest of the assignment outlines one particular method you may try to implement, but you are welcome to be creative. Please answer all the attached questions as best you can, regardless of method you use.

An Approach:

- **Generate Representative Colors:** Given the color data in **color.csv**, generate a family of 'representative' colors that reflects the true colors as accurately as you can. Different shades of green may be lumped together into a single representative shade. The color data can then be reduced, by replacing every color with its 'representative' color - the way you might replace handwritten letters with typed characters, rendering the diversity of handwritten lettering more readable via uniformity.
- **Generate Representative Patches:** Given the gray patch data in **input.csv**, generate a family of 'representative' patches that reflects the patch data as accurately as you can. Patches can be thought of as jigsaw pieces that fit together to make the whole image - how many types of jigsaw piece do you actually need to construct the whole, or something close enough to it? The patch data can then be reduced, replacing each patch with its representative.
- **Determine a Representative-to-Representative Map:** The data in **color.csv** and **input.csv** can then be converted to an association between representative patches and their representative colors. For every representative patch, determine from this data what the best representative color to associate with it will be.
- **Colorize:** For any new gray image, any 3x3 pixel patch can be colored by first a) determining its patch-representative, and then b) identifying the color-representative associated with that patch.

Questions:

1) Taking the shade values to be integers between 0 and 255, how many possible color values are there? How many possible grays? How many possible 3x3 gray pixel patches? Should a 3x3 pixel patch be enough information to guess the center color, or not enough? Under what contexts might it be enough to try to associate every shade of gray with a specific color, and color an image on a pixel by pixel basis?
2) Why might it be a good idea to reduce the color-space or the patch-space, for instance by lumping certain shades of green together into one color? What should a good coloring algorithm do if it was fairly confident a pixel should be green or red, but certainly not blue?
3) Describe your algorithm, and the design or parameter choices that went into your algorithm. What was your reasoning? What were the difficulties you had to overcome?

4) How can you use the provided data to evaluate how good your coloring algorithm is? What kind of validation can you do? Quantify your algorithm's performance as best you can.

5) Once you have trained your algorithm on **input.csv** and **color.csv**, use it to provide colors for each of the example patch data in **data.csv**. Submit the computed colors in a **csv** file, each row corresponding to the color of the corresponding row in **data.csv**.

6) Where does your algorithm perform well, where does it perform poorly? How could you (with greater time and computational resources) improve its performance? What parameters or design decisions could you tweak? Would more data on more diverse images help or hinder your algorithm?

7) Do you think this is a reasonable approach to the problem? What would you change?

8) What would happen if you tried to color a grayscale image of a tiger, after training your algorithm on pictures of palm trees?

Bonus) Write your program to take in an image, convert it to grayscale, train your algorithm, and then use the algorithm to color example grayscale images. How does your algorithm do, visually?